Analyzing BFT & Proposer-Promised Preconfirmations

In a previous post, we introduced the Espresso Sequencer: a decentralized, highly-performant shared sequencing platform that reduces the complexity in achieving atomicity, composability, and interoperability between rollups. The Espresso Sequencer is an external, opt-in protocol run by L1 validators. As part of our protocol, the Espresso Sequencer natively offers BFT preconfirmations to rollup users. BFT preconfirmations are backed by the security and liveness guarantees of a BFT consensus algorithm such as Espresso's HotShot. In this post we will analyze another external, opt-in preconfirmation protocol: proposer-promised ("PP") preconfirmations.

Architecture of Proposer-Promised Preconfirmations

Proposer-promised preconfirmations were initially introduced as "based preconfirmations." Unlike BFT preconfirmations that require agreement among the entire validator set, PP preconfirmations only require approval from individual validators (namely, from "proposers" of the underlying sequencing protocol). The protocol description below reflects our understanding of the PP preconfirmation paradigm, fleshing out many details necessary to address in order to materialize the original proposal. The only assumption made about the underlying sequencing protocol is that it operates via a rotation of proposers.

Validators opting to offer PP preconfirmations are known as *preconfers*. By opting in, preconfers agree to participate in an externally-managed protocol where they can be slashed if they do not honor their preconfirmation promises. Each rollup can customize the PP preconfirmation ordering and slashing conditions to meet their individual needs. Preconfers are ordered by their next proposer slot in the sequencing protocol and can offer a spectrum of preconfirmation types, ranging from strict promises of transaction execution against a particular state root to weaker promises of transaction inclusion only. Certain types of preconfirmations, such as execution promises against a specified state root or intent-based execution promises, can only be offered by the very next preconfer since only they have control over the latest rollup state. Other forms of preconfirmations, such as ordering-only or inclusion promises, can be offered by any preconfer. Some rollups may decide to only allow certain types of preconfirmations. Otherwise, it is up to preconfers to decide which types of preconfirmations to offer[^1].

Exchange between users and preconfers

Obtaining a PP preconfirmation is performed in three steps:

Step 1: Sending a preconfirmation request. A user sends a *preconfirmation request* off-chain to one of the next preconfers in the proposer ordering. At minimum, the preconfer request must include the user's transactions and a preconfirmation fee. As we describe in a previous post, the transactions and fee can be atomically bundled such that the fee is not valid without inclusion of the user's transactions. This prevents an attack where a preconfer rejects the user's request but still includes the user's fee payment in their block. The bundle can be further expanded with additional validity data, such as execution conditions or the intended preconfer's public key. Depending on the type of promise the user is requesting, they may also need to include additional data in their request such as the latest state root or sequencing number seen from the preconfer.

Users may choose to send preconfirmation requests to multiple preconfers in case the first preconfer misses their slot, raising the need to handle double-promises of the same transaction. This means there should be a mechanism to void the request fee if a transaction is executed in an earlier block[^2].

Step 2: Approval/Rejection of the preconfer request. Next, the preconfer receives the preconfirmation request from the user and must decide whether to return a promise or reject the request. The preconfer must verify the preconfirmation request is valid, and whether the preconfirmation fee is sufficient. We discuss the difficulty of determining fee sufficiency later on. If the preconfer decides to accept the request, they return a slashable promise back to the user. This promise at minimum consists of a signature over the user's request. Depending on the type of promise offered, the preconfer may need to include additional data in the promise such as an updated state root or sequencing number. They may also need to publicly stream this data so other users can make future preconfirmation requests.

Step 3: User verification of the preconfirmation. Finally, the user receives and verifies the promise. At minimum, the user verifies the preconfer's signature. Depending on the type of promise, the user may also need to use additional data from the preconfer to verify that their request conditions were met. If this verification fails the user can submit the promise for slashing.

The three steps above constitute a binding agreement between the user and preconfer. Bundling transactions and preconfirmation fees such that one is not valid without the other removes the need for an additional fair exchange protocol between the user and preconfer. Later on, we address what may happen if the preconfer does not respond with a promise in a timely manner.

Inclusion and Execution

PP preconfirmations are first included in a sequencing block, and then executed, as described below.

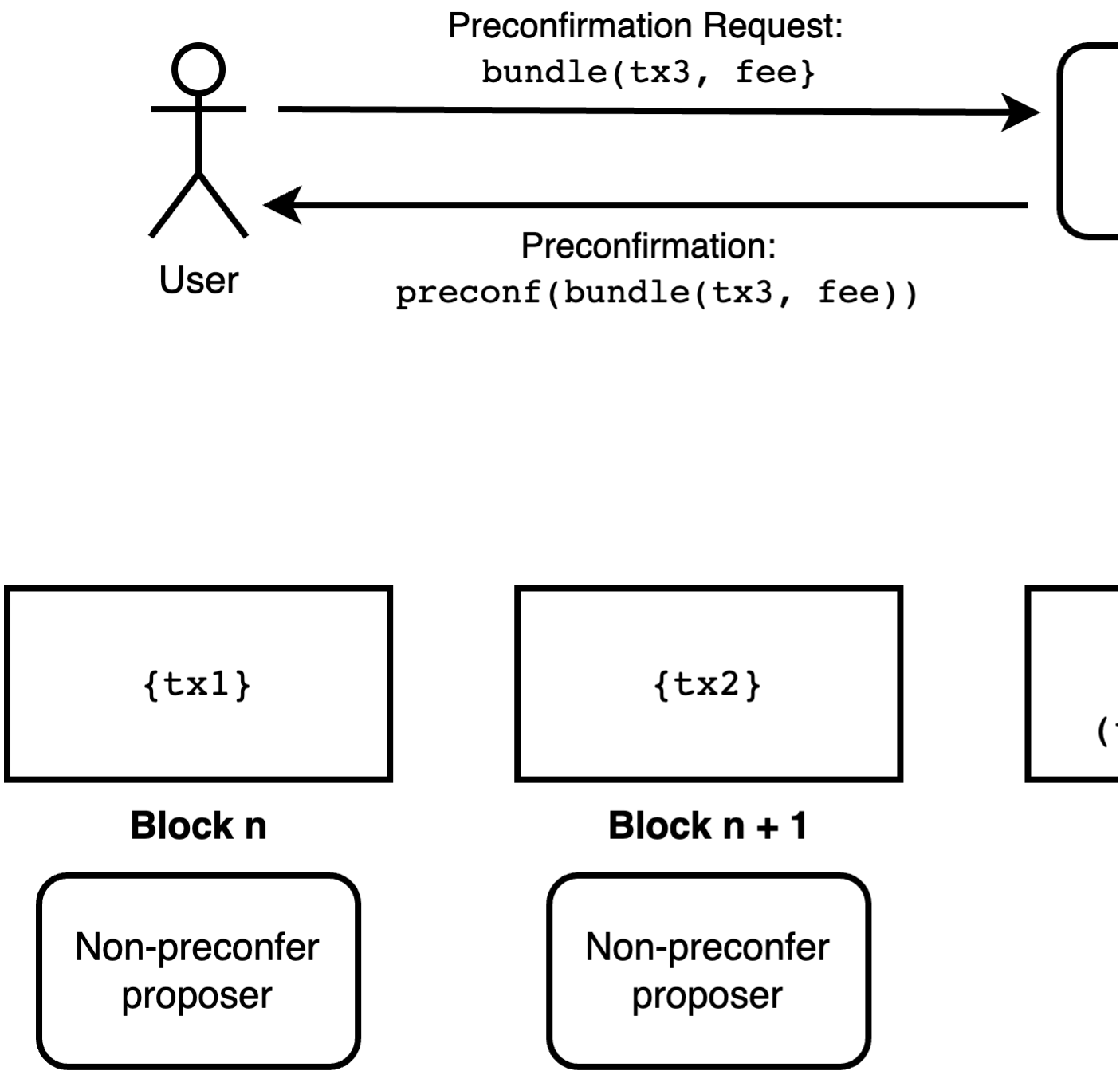

Figure 1: PP Preconfirmations Protocol Flow

The next preconfer in the proposer ordering is not necessarily the next block proposer of the sequencing protocol. Without loss of generality, let's assume proposers are chosen uniformly. If $\frac{1}{\rho}$ sequencing validators opt to become preconfers, it is expected that a preconfer will be a proposer on average every ρ blocks. Preconfirmed and non-preconfirmed transactions can be included in any sequencing block, but all transactions are queued for execution until a preconfer successfully proposes a block[^3].

Non-preconfers may include whichever transactions they wish in their blocks, including transactions preconfirmed by other validators. Non-preconfers do not need to be aware of the PP preconfirmation protocol.

Preconfers may also include whichever transactions they wish in their blocks so long as their preconfirmation promises are met. Preconfers must include any preconfirmations they made unless one of the following conditions applies:

- The preconfirmed transaction was nullified and would not have been executed. This could occur if a conflicting transaction executed in an earlier block. Examples of conflicting transactions could be: a different transaction with the same nonce, the same transaction that was unbundled by the user and submitted as a non-preconfirmed transaction, or a transaction included by an earlier preconfer. The preconfer may need to participate in arbitration to prove that the transaction was properly nullified.
- The preconfirmed transaction was included in a previous block. In this case the derivation pipeline will take care of correctly inserting this transaction into the preconfirmed transaction ordering for the next rollup block. In practice, there will likely be a practical limit to how many blocks back a preconfirmed transaction can appear before its preconfer's block.

Once the preconfer's block is sequenced by the network, a rollup's derivation pipeline, which we describe in a previous post, combines blocks from after the last preconfer's block through the current preconfer's block to form the next canonical *rollup* block. The derivation pipeline can be customized for each rollup to enforce certain ordering policies while still honoring preconfirmations. One particular ordering policy for the derivation pipeline would be as follows: order all non-promised transactions after ones that have preconfirmation promises, and among preconfirmed transactions, first order by ordering-promises (if used), then by the order they appear in blocks (both preconfer or regular blocks). It is required that a rollup's derivation pipeline ordering policy is deterministic for all scenarios. Precisely defining this policy may be a non-trivial task. Any rollup participating in PP preconfirmations will need to update their current derivation pipeline to handle a new ordering policy.

Once the derivation pipeline derives the canonical block, the block is executed.

Discussion of Proposer-Promised Preconfirmations

PP preconfirmations is a nascent proposal. Its interaction with ecosystem players and MEV factors is not fully understood. This exploration aims to illuminate several directions in which it may evolve.

Are preconfers incentivized to provide fast preconfirmations?

The primary motivation of the original PP preconfirmation proposal was to provide users with fast preconfirmations at the speed of the network (on the order of 100ms).

Proposers in any sequencing protocol are incentivized to propose the most valuable block they can. Often, as in the case of Ethereum today, proposers auction off their building duties to designated "builders" that optimize for finding the maximum extractable value (MEV) from transactions in order to build the most valuable block. To build the most valuable blocks, building entities solve an *offline* MEV-maximization problem. In the offline MEV problem, builders are able to consider the entire set of possible transactions at once and output the entire block only one time. If a builder learns of new transactions before the deadline to submit their block, they can add these new transactions to the set of all possible transactions, and rebuild the block as a whole.

Preconfers, however, must solve a slightly different, *online* MEV maximization problem. Preconfers can receive preconfirmation requests at any time before their slot. At the time they receive a request, they likely do not know all the transactions that may be included in their block - they could receive future preconfirmation requests and additional non-preconfirmed transactions between the receipt of the current request and their block proposal deadline. Thus preconfers must decide whether to preconfirm a request for the given fee without the complete information needed to determine if ordering that request is the MEV-maximizing decision over the block as a whole. Preconfers must incrementally build their blocks and repeatedly solve this online MEV-maximization problem with every promise they commit to in order to determine if the user's fee is adequate.

It is therefore plausible that preconfers may choose two behaviors: Preconfers might decide to offer inclusion-only promises to users, instead of stricter ordering or execution promises. This way they can avoid the repeated online MEV problem in favor of a simpler, offline MEV-maximization problem. Preconfers may also choose to delay sending their preconfirmation promises to users so that they have more time to evaluate the online MEV problem[^4].

One potential solution is to allow users to "expire" their preconfirmation requests if preconfers do not respond in a timely manner. However, it is likely the user will want to submit their transactions at a later point, and preconfers can use this knowledge to potentially frontrun the user. Alternatively, it is possible preconfers may be incentivized to offer fast preconfirmations so that they can collect the maximum amount of fees during their slot. However, this still requires them to evaluate the online MEV problem at some level, and it is unclear which behavior preconfers will choose.

What is being auctioned?

Regardless of the particular ordering policy chosen by rollups, there may be multiple opportunities for proposers of the underlying sequencing protocol to interact with builders and auction block space in ways that impact rollup transaction ordering. Markets may evolve around any such opportunity in ways that are too early to predict, including for example: when builders request inclusion-promises for transactions from preconfers, when they bundle promised-only transactions in the blocks leading up to their preconfer block, when they request ordering-promises for transactions from preconfers, and as usual, when they bundle non-promised transactions in blocks. It is also likely that preconfers themselves will auction their preconfer rights to more sophisticated builder entities.

Preconfers solving MEV-maximization problems require a certain level of sophistication. Thus, it is likely that just as PBS emerged in Ethereum, a similar preconfer builder separation will emerge. Finally, users must also calculate the online MEV problem in order to know the appropriate fee to send the preconfer. Users could solve this themselves, which may be quite difficult, or they can outsource the pricing to a trusted relay or the preconfer themselves.

Do PP preconfirmations improve user experience?

Although users may *potentially* receive fast preconfirmations, they can experience increased delay on transaction execution. As mentioned above, preconfers are proposers of the sequencing protocol on average every ρ blocks, meaning most transactions, preconfirmed or otherwise, will have to wait ρ blocks for execution. It is an open question as to what proportion of proposers will opt to preconfirm for particular rollups. Users may face worse latency to discover their transaction execution status with PP preconfirmations than they do today. Furthermore, users that choose to not acquire preconfirmations can be more easily frontrun by preconfirmed transactions. If a high number of L1 validators opt in to preconfirming, and a high number of users find preconfirmations useful, the negative impact on rollup latency may be lessened, but additional latency is still present since the rollup derivation pipeline cannot start until after an L1 finality event. It is too early to say which direction this will evolve in.

Depending on which types of preconfirmations are offered, preconfers may only be able to process a single user request at a time. In the case of strict execution promises, for example, preconfers must publicly stream the latest execution state so that users know which state their transactions will be executed against when requesting a promise. Thus, in times of congestion it may be difficult for users to keep up with the latest data being streamed from preconfers, thus making it difficult for them to successfully request stricter forms of preconfirmations. It is possible the rollup-specific builders would bundle requests, but then the benefits of obtaining fast preconfirmations may be reduced. In any case, these forms of strict preconfirmations favor more sophisticated users or users located close to preconfers.

As mentioned above, each rollup defines its own PP preconfirmation ordering policies, slashing conditions, and allowed types of preconfirmations. Validators opt in to preconfirming on a per-rollup basis, and may choose to only offer a subset of preconfirmation types. Thus, users looking for particular preconfirmation types or cross-rollup guarantees may have difficulty finding a preconfer that will honor their wishes. As mentioned above, it is unclear whether preconfers will be incentivized to offer stricter, execution-type promises as it requires solving the online MEV problem. It is also unclear if current PBS builders that offer execution promises will still be able to offer them in the new PP preconfirmation model. It is possible that while a preconfer is a proposer on average every ρ blocks, preconfers offering the services a user wants are proposers more rarely. One potential, incomplete solution would be for rollups to form preconfirming coalitions where several rollups agree to unify their PP preconfirmation designs so that validators can opt into preconfirming for multiple rollups at once.

Finally, PP preconfirmed transactions are not final; it is possible for a preconfer to honor all their preconfirmations but have their block reorganized in the chain, thus breaking preconfirmation guarantees. In this case, preconfer slashing doesn't help since it is unfair to slash preconfers for reorganizations that are out of their control.

BFT preconfirmations and PP preconfirmations

Both BFT rollup-sequencing and PP rollup-sequencing run external protocols that L1 validators can opt into, and both require an integration from rollups. However, they operate differently and provide rollups and users different tradeoffs.

A key design choice of PP preconfirmations is to treat the underlying sequencing protocol as a black box in order to reduce the complexity of implementation. PP preconfirmations aim to provide fast preconfirmations by having preconfers act as a slashable centralized sequencers during their preconfer slot. So long as the single preconfer is honest, a user's preconfirmation will be honored. BFT preconfirmations, on the other hand, take a different approach by using the internal details of a BFT consensus protocol to offer different preconfirmation guarantees. While more complex to implement, this approach has several key benefits:

- BFT sequencing, through which BFT preconfirmations are natively offered, leads to faster rollup pipelines. While the promises obtained through PP preconfirmations may potentially be faster than BFT preconfirmations (assuming the concerns we raised in our analysis are resolved), rollups using BFT sequencing may begin their derivation pipeline immediately after receiving the committed output of the consensus protocol. Rollups using PP preconfirmations, on the other hand, must wait until the preconfer's block is both proposed (which may take several blocks) and made final on the sequencing protocol before starting the derivation pipeline. This can introduce a non-negligible delay in the rollup production flow, especially since not all validators opt-in as preconfers.
- BFT preconfirmations are backed by the economic security of the entire Espresso Sequencing network, which consists of a subset of L1 validators, not the security of a single validator. Greater than $\frac{1}{3}$ of network stake must be compromised for BFT preconfirmations to not be honored, compared to only the collateral of a single validator in PP preconfirmations.

Finally, BFT preconfirmations and PP preconfirmations are entirely composable. Both protocols can be run in tandem to provide users with different levels of preconfirmation guarantees, and in both, builders have new opportunities to interact with validators and bid for block space.

[^1]: Preconfers will need to publicly advertise which preconfirmation services they offer. It is possible a marketplace could form that aggregates this information for users.

[^2]: Duplicate transactions will be automatically voided since they contain the same transaction nonce. However, it is likely that a preconfer does not know that a previous preconfer has agreed to include a user's transactions in their block at the time they make a preconfirmation promise. Thus a future preconfer could lose fees from transactions they assumed would be valid when they agreed to include them. It is an open question how to address this.

[^3]: Depending on a rollup's particular ordering policies, execution-promised preconfirmations may be executed immediately without waiting for the next preconfer block if 1.) the execution promise is the first promise given by a preconfer for that slot, or 2.) all execution promises preceding this promise have already been executed.

[^4]: Preconfers may wait to return a promise until they receive many execution requests for a particular state root so that they can choose the one which offers the highest fee or which one creates other MEV opportunities. For example, a preconfer may delay accepting a promise request so they can potentially receive another transaction that can be backrun with the preconfer's own transactions.